Architecture Query Language Framework

Bartosz Michalik
DistriNet Labs, Departement of Computer Science, KULeuven
Email: bartosz.michalik@cs.kuleuven.be
Danny Weyns (Advisor)
DistriNet Labs, Departement of Computer Science, KULeuven
Email: danny.weyns@cs.kuleuven.be

***Abstract*—Software architecture is a key factor in the success of software product line (SPL) engineering. A SPL architecture has to incorporate the commonalities as well as the variabilities of the products in a SPL. Presenting variability in the views and models that suite the stakeholders' concerns is essential for the success of a SPL. However, a literature study has taught us that current research provides insufficient attention to incorporate variability in the architecture description. Moreover, experiences with stakeholders in the field reveal that there is a need for flexible generation of architecture views tailored to the specific needs of the stakeholders of a SPL.**

To tackle these challenges we propose the Architecture Query Language Framework. Central in this framework are an architecture model for SPL, Dynamic Viewpoints (DV), and an Architecture Query Language (AQL). The architecture model supports modeling of domain-specific architectures, covering structural, allocation and variability concerns. The model is supported by an architecture repository that reifies architectural knowledge of the SPL. A DV defines the data model to support on-demand view derivation from the architecture repository according to the stakeholder needs. DV are supported by an Architecture Query Language and an accompanying engine which allows flexible and fine-grained querying of the model repository. The research will be evaluated in two case studies with industrial partners and an empirical study in the context of master course on advanced software architecture.

I. Motivation

Software Product Lines (SPL) are receiving increasing attention in software engineering research. SPL are a key enabler to establish strategic reuse. In a SPL approach, products are derived from the common set of reusable assets defined in the context of a specific domain [27].

One of the key factors in the success of SPL is the SPL architecture. Software architecture defines a system's essential structures, which comprise software elements, the externally visible properties of those elements and the relationships between them [4]. The architectural description of a SPL architecture should include the definition of the commonalities (platform) and variabilities of the SPL. Although, numerous variability models have been presented, the specification of variability at the architectural level seems to be insufficient.

To document a software architecture, multiple architecture documentation frameworks have been proposed (e.g. viewpoints based [18], [4] and design decision centric [17]). Architecture documentation is a key communication vehicle and as such should be expressive enough to describe the various concerns of the stakeholders. These concerns are usually captured using different models and views. Architecture addresses the most important requirements and is typically expressed with the use of the stakeholders' domain concepts. Presenting software architecture and variability in the terms and concepts of the stakeholder is essential for the success of a SPL [25]. Work on architectural views that consider variability exists, e.g. [31], [30]. However, the mapping between variability models and architecture views needs further study.

As software evolves over time, architecture and all architecture related artefact (which document the stakeholders' concerns) should be maintained to keep the architectural documentation consistent. Whenever an artefact from the platform is changed, the changes can be propagated to all products in which the artefact is used. Without proper tool support, consistency management of architecture documentation can be a laborious task.

In this research, we propose the Architecture Query Language Framework that aims to contribute to the challenges discussed above. This framework allows on-demand view derivation from an architecture repository. This repository is based on an architecture model with first-class support for variability that allows specification of domain-specific SPL architectures. The approach is supported by an Architecture Query Language and accompanying engine which allows flexible and fine-grained querying of the architecture repository. The research will be evaluated in two case studies with industrial partners and an empirical study in the context of master course on software architecture.

II. Objectives

The overall goal of this research is:

> *To define a flexible Architectural Query Language (AQL) which can be used to dynamically derive views from a SPL architecture repository.*

To realize the main goal, two additional goals were defined:

- To define an architecture description model (the data model used by the AQL) with support for structural and deployment elements;
- To support SPL variability modeling at the architectural level; i.e. incorporate variability in the architecture description model.

To realize these goals multiple tasks were defined:

A. Theoretical fundamentals; i.e. the study, definition and development of:

- Domain-specific architecture models (specific syntax and semantics);
- A generalized architectural model (generic syntax and semantics);
- Variability models and their integration with the architectural model;
- The relations between different model types.

B. Validation of the fundamentals, i.e. :

- Development of the necessary tools to put the fundamentals in practice;
- Application of the approach in two different application domains;
- Empirical evaluation of the approach with advanced master students.

An overview of the concrete planning is presented in section VI.

III. Research methods

This research aims to realize relevant, evaluated and scientifically validated results in the domain of software architecture for SPL. The scientific methodology is based on four pillars:

1) *Application-driven research*. This research starts from real-world requirements and develops solutions relevant for the cooperating parties. The intermediate findings are evaluated in practice, providing feedback to steer the basic research.
2) *Iterative approach for complexity management*. This approach allows focusing on specific sub-problems and gradually integrating them to build the solution for the overall problem.
3) *Empirical evaluation of results*. An empirical study will be set up to evaluate the research results. This allows to obtain qualitative insights and feedback from practitioners.
4) *Validation for selected domains*. The research results will be applied and validated in two SPL case studies. These studies in collaboration with industry will demonstrate the feasibility and use of the generalized architecture model and the AQL in two different settings.

The research plan comprises the following stages:

- Domain analysis and definition of domain-specific models (definition of domain concepts and variability);
- Definition of the generic architecture model through analysis of domain specific models (definition of generic and domain-specific elements and relationships)
- Domain-specific viewpoints definition (along with domain specific query primitives and mechanisms);
- Definition of the architecture query language (incl. realization of the query engine).

IV. Sketch of the proposed solution

To address the need of deriving customized architecture views, we propose the Architectural Query Language Framework. Figure 1 shows the realization overview of the AQL Framework.

The *Views derive component* provides on-demand views. To build a view, the component uses Dynamic Viewpoint (DV) definitions, which are stored in the *Viewpoints repository*, and architectural data, collected from *Models Repository*. A DV defines a data model and data presentation method to address one or more concerns of a given set of stakeholders. Viewpoints can be defined by the architect or other stakeholders.

To actually access the architectural data, the *Query Engine* component is used. The query engine interprets and executes the queries. Queries are defined using the Architecture Query Language (AQL).

Whereas a viewpoint provides a description of the conventions and the data model for the realization of a view, the AQL provides fine-grained primitives to query the architecture model to generate the view according to the given viewpoint definition.

The architecture model repository for a given system can be populated in two complementary ways:

- Harvesting process (*Harvesters component*) - which is an automatic process of collecting the architectural knowledge from existing artifacts. Harvesting is particularly useful when architecture data has to be discovered from existing/legacy systems.
- Manual update (*Modeling tools* component) - the models are updated by stakeholders (typically the software architect); the repository model can be updated with data manipulation tools provided for the specific description language used by stakeholders.

Queries defined in the AQL are used for retrieving the data from the architectural repository. The structure of the repository is defined by *Architectural repository model*, see Figure 2.

Model concepts are defined at two levels: domain-specific and generic. The *Generic model* introduces concepts for expressing the allocation of software elements to hardware nodes (*Allocation* model), structural elements and relationships between the elements (*C&C* model), and variability concerns (*Variability* model). In addition, the relations between concepts from different models are part of the generic model.

Domain-specific concepts represent the elements and relationships between elements for a particular domain. Domain-specific concepts are defined in terms of the generic model concepts. As such, the generic model provides core architectural knowledge which can be reused to describe domain-specific model concepts. Besides reusability, the two model levels also supports fine-grained expression of queries of the AQL. In particular, it allows a flexible combination of generic language constructs with domain-specific constructs.

The complete AQL Framework will be defined and developed in a bottom-up approach. Subsequently, we will apply:

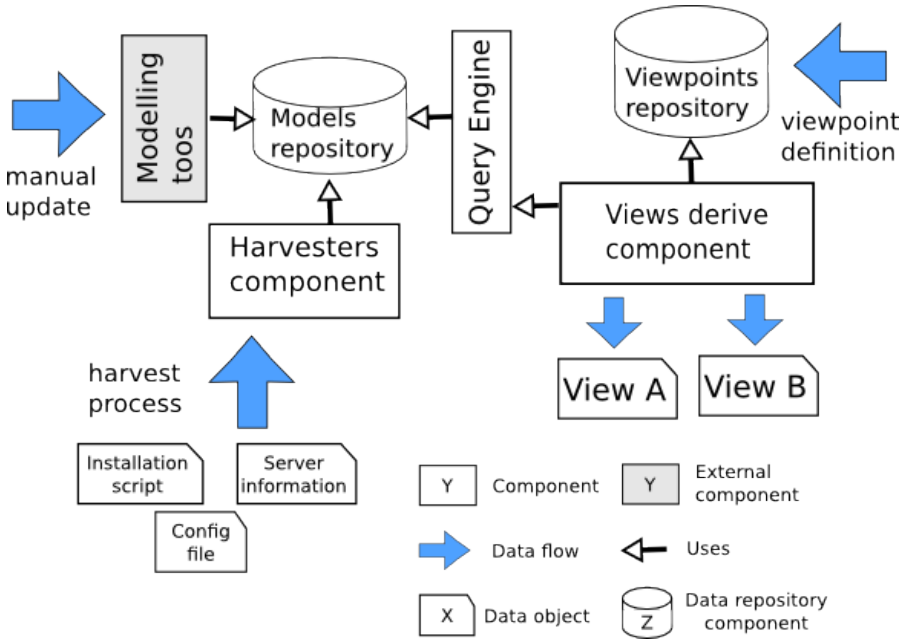

Fig. 1. Realization overview of the solution

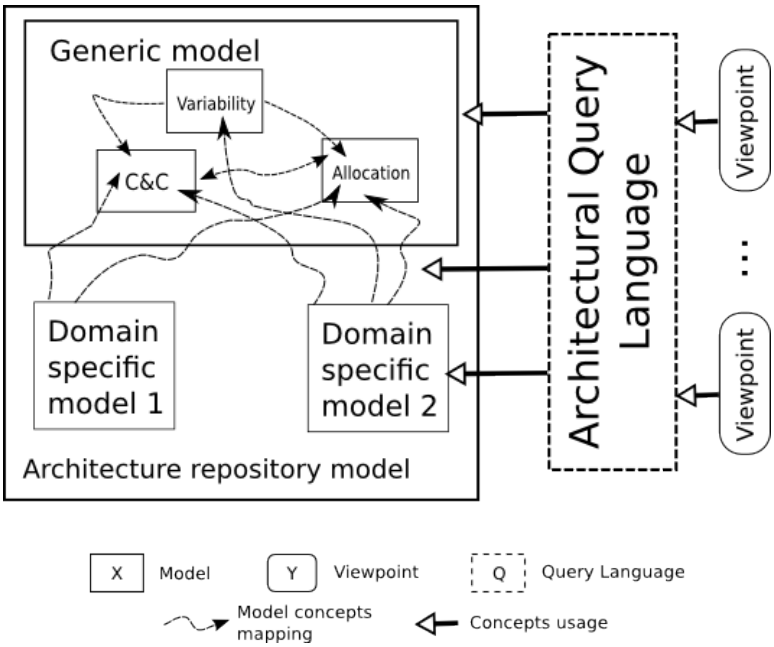

Fig. 2. Conceptual overview of the solution

1) Analysis of two domain architectures, including variability analysis;
2) Definition of domain-specific models and tools (definition of Dynamic Viewpoints and AQL queries for the domains);
3) Generalization of domain-specific concepts and tool support (definition of reusable parts of viewpoints - in particular a viewpoint for variability - and definition of the grammar of the AQL).

In the remainder of this section, we will elaborate on the main elements of the AQL Framework.

A. Case studies for domain-specific models

This research is driven by two industrial case studies. For each case, domain-specific models are defined.

1) Egemin Warehouse Inc.: [1] builds logistic systems for transporting goods in warehouses or factories. The systems are developed in the .NET environment. An overview of a system

[1]As the real names and labels may not be disclosed, we use substitutes.

is shown in Figure 3. *Client* and *Service* components communicate using the *Logistic Platform* (SPL platform) which is a distributed middleware component. Host connections are omitted to simplify the diagram. Each *Component* consists of several *Assemblies* which are reusable deployment units.

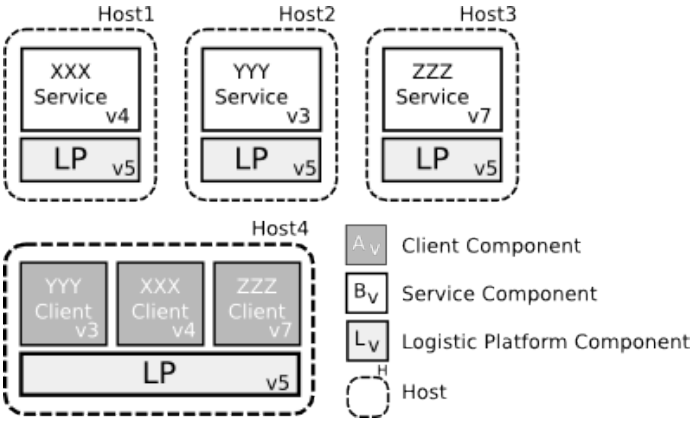

Fig. 3. Overview of Egemin Warehouse Inc. Installation. X_{vY} stands for element *X* in version *Y*

Since logistic systems are long-lived systems, the SPL platform as well as specific product-specific are subject to evolution. However, upgrading even a single component in a logistic system is less trivial as it seems. Identifying the set of assemblies that have to be replaced is essential to prevent unnecessary shutdown of system components and unnecessary recertification costs. However, determining the scope of an upgrade is hard. Assemblies are strongly interdependent, so upgrading one component of the system may require other components to be taken offline for an upgrade as well. Even for a simple installation, the number of dependencies between the assemblies can be in the order of 100. Updating the SPL platform requires an impact analysis of every affected installation. Manual analysis is error-prone and therefore a tool supported process is desirable. The AQL Framework aims to semi-automate the updating process of Egemin's SPL.

The Egemin case study is an example of project-integrating adaptation scenario [28]. The identification and codification of the variability at the architectural level is an important concern which must be solved.

2) SDG Bluetooth Advertisement: SDG is a company that provides mobile advertising solutions. Advertisements are geographically localized (with the use of Bluetooth technology) and add support for interactive capabilities to traditional advertisement channels (e.g. billboards).

The Bluetooth Advertisement system is a bluetooth broadcasting server which support multi-content transfers. The existing solution is well defined, but provides a limited set of functions. To enhance its market position, the company is planning to introduce an end-customer configurable family of the advertisement products.

An analysis of the existing product architecture has shown that the current system is not flexible enough to support the intended change. Therefore, a new SPL architecture is being designed with explicit support for variability.

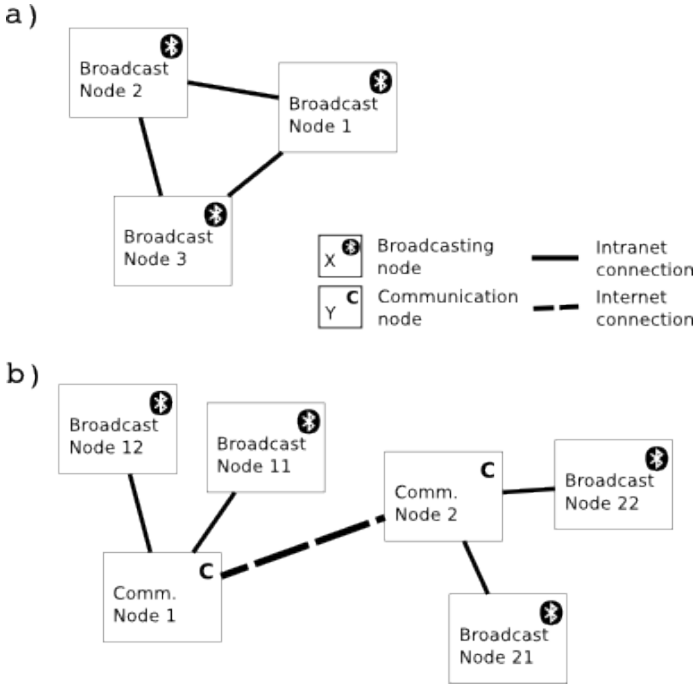

Fig. 4. Overview of SDG Bluetooth Advertisement configuration scenarios. In diagram *a)* the single-location-multi-node configuration, and in *b)* multi-location-multi-node are presented.

Figure 4 presents two scenarios of the planned product configurations. In the part *a)*, a single-location-multiple-nodes scenario is shown. This configuration enables product installations in large facilities (e.g. shopping malls). Part *b)* presents a multi-location installation scenario. In this scenario, different locations share synchronized information about the state of the advertisement campaign (e.g. promotion code support in a chain of restaurants).

The variability in the Bluetooth Advertisement solution can be observed at the level of content delivery strategies, nodes functionality, and installation configuration. The product is planned to be build on top of the OSGi platform.

The introduction of the SDG SPL can be considered as an independent adaptation scenario [28].

B. Generic model

The *Generic model* (see Figure 2) contains the concepts required for Allocation and Components & Connectors(C&C) models [10], and variability [27] models. The C&C model allows to express execution units and interaction mechanisms between them. The Allocation model captures the allocation of software deployment elements to infrastructure elements. In addition, the generic model includes the relationships between elements of Allocation and Components C&C models (e.g. the relationship between software components and deployment units). The Variability model consists of two parts:

- Features model - which forms a separate sub-model of the generic model;

- Variation points and variants - which are directly introduced to Allocation and C&C models.

A *feature* describes the variability on the functional level (i.e. *what* is realized). Each feature maps to one or more *variation points* which have to be realized to introduce the feature in a given product. A variability point defines the policy of its realization (i.e. *how* it is realized). A *variant* realizes one of more variation points. As such, relations between variants and features are inherent parts of the generic model. Our approach to variability modeling in architecture is inspired by the model proposed in [27].

A SPL architecture can be seen as a platform (definition of the commonalities) along with the assets (variants) database. Therefore, initial *product architectures* can be derived from the SPL architecture by features selection. Feature selection defines the customization of a set of variability points. This set (along with the features and variants relations) can be used for the automatic pre-selection of suitable variants.

The concepts of the generic model can be used to define domain-specific models. The architectural data repository reifies the generic model and its domain-specific extensions to store architectural knowledge of concrete systems. The generic model supports the common parts of the AQL grammar, and the general parts of the AQL query engine and viewers that are used to present views to the stakeholders.

C. Relations between domain-specific and generic models

A domain-specific modeling concepts are expressed in terms of generic modeling concepts. Different types of relations can be distinguished, i.e.:

- Elements equivalence mapping (for example in the Egemin case: Assembly *is a* Deployment Unit)
- Rule base mapping (for example in the SDG Bluetooth Advertisement case: OSGi Bundle *is a* Component *having* Manifest artifact *of type* Resource)

D. Architecture Query Language

The Architectural Query Language defines a grammar for the specification of queries. The queries are interpreted and executed by the query engine to derive architectural data stored in repository. The repository reifies the concepts of the generic model (as well as the related domain-specific concepts). This allows for the reuse of querying mechanism. However, the semantics of the generic model concepts can be hard to understand by stakeholders. Therefore, the AQL definition will provide a domain-specific, user-customizable interface to architecture knowledge repository. Under the hood, the AQL exploits the mappings between domain-specific and generic model concepts.

Besides retrieving views from the architecture repository, other usage scenarios of the AQL are possible:

- Data mining - a possible stakeholder's concern in this scenario is historical data analysis. For this scenario, the AQL queries can be executed for multiple projects (e.g. find all products which containing finger print reader variant of the door lock).
- Product derivation for a SPL - in this scenario, the AQL queries can extract possible variants for sets of features that (along with the platform) define a product architecture.
- Patterns/concept discovery - querying one domain repository with queries defined for another domain can help in discovering patterns common to both domains.

E. Dynamic viewpoints

A viewpoint is defined as the "work product establishing the conventions for the construction, interpretation and use of architecture views and associated architecture models" [16]. In other words, a viewpoint is the definition of the view. A Dynamic viewpoint is the definition of a view which can be derived from existing architectural models/knowledge. Dynamic views can be used to present up-to-date architectural information about the concern of a given stakeholder (or set of stakeholders).

The definition of a viewpoint consist of:

- A description of the purposes and audience for the view (*Why* is the viewpoint defined).
- A set of AQL queries - an AQL query defines the data model required for a given view derivation. This model is used to retrieve a selection of the repository content (and in this sense created on-demand). The set of queries defines the data available for a given view (*What* data is presented).
- A viewer - defines the presentation of the architectural data of the view (*How* is the data presented).

Views can be used to present single-model information or inter-model information. The AQL allows for even more advanced queries with the use of object attributes and structural dependencies. Examples of viewpoints which can be defined by stakeholder are: component-deployment viewpoint (defines the relations between allocation and C&C elements), security variants viewpoint (defines the variability of a product filtered for a specific concern), and conflicting variation points viewpoint (useful for analysis purposes).

Viewers and AQL queries have the potential to be re-usable artifacts for the definition of multiple viewpoints. In the AQL framework, the effort required for view maintenance is only the cost of maintaining the viewpoint definition. In this way views can be considered dynamic - the viewer presents an up-to-date snapshot of architectural data, and its content evolves with the evolution of the system and its model.

V. Expected contribution

The first expected contribution of the this research is improved understanding and expression of variability at the architectural level. Since this research is conducted in collaboration with industrial partners, concrete needs for variability modeling have to be identified and tackled. The practical use of the AQL Framework can result in the identification of new interesting scenarios of SPL variability analysis and product derivation.

Second, the AQL Framework aims to make a step towards providing tools for flexible articulation of stakeholders concerns. The definition of dynamic viewpoints can bridge the gap between architectural knowledge (internal - knowledge management) and representation (external - expression of concerns). In other words this research underpins a compelling motivation for reuse of architectural knowledge-entities relations, and architectural knowledge reification in general.

Third, the AQL Framework has the potential to reduce the cost of maintaining architecture knowledge. As dynamic views can be derived on-demand, they are implicitly consistent with the artifacts representing the system's software architecture.

Fourth, the effort of defining a multi-facet generic model for SPL architectures can facilitate the understanding of domain-specific architectural concepts. The mapping of domain-specific architectural concepts to general concepts can contribute to a better understanding of architectural concepts/patterns common to a particular domain (e.g. the architectural concepts for the .NET environment).

This project aims to define a flexible AQL Framework for the generation of dynamic views that is applicable to a variety of application domains. However, the generality of the approach may be restricted by the specificity of the case studies that will be studied in this work. Nevertheless, there are good indications that this risk is limited. First, the research is conducted with industrial partners which deliver usable solutions for different application domains. The generalization over these domains will contribute to the applicability of the approach. Second, the additional empirical evaluation will provide feedback that can be used to adjust the approach with respect to identified reusability problems.

VI. Progress in solving the stated problem

As explained in section III, this research has a multi-stage character. Initial ideas have been presented in [24]. The ongoing work focuses on domain-specific tasks in the two case studies.

An overview of the research planning is shown in Figure 5. The tasks are described with respect to application domains. The Egemin case is divided into two phases. In the first phase, the evolution of Egemin products is studied, including the definition of domain-specific models, queries and views. Then, a prototype tool is built supporting dynamic evolution of logistic systems. In the second phase, modeling of the platform's variability is introduced to the domain model (and the tool). After each phase feasibility evaluation is performed. The SDG OSGi case has a similar structure.

Recently a tool to support the software evolution of Egemin's SPL was developed. Figure 6 shows a snapshot of the GUI of the tool with an evolution view. This view provides information about the differences between the actual and expected installation of a product. Engineers and maintainers at Egemin use the view for *change impact analysis* (e.g. how do updates of common platform components influences the products) and *consistency analysis* (e.g. are all the dependencies of the updated installation resolved). The architecture repository model used in the evolution viewpoint is domain-specific. The underlying data model was built with Eclipse Modeling Framework (EMF) technology[2]. The access to the data collected in the repository (using harvesters) was realized by a query engine. This prototype query engine was built with the EMF Query module. Figure 6 shows an excerpt of an on-demand view. The evolution viewpoint defines a set of queries for data extraction, a tabular based interactive viewer, and data binding. However, this is a proof of concept solution and a more flexible mechanisms will be implemented in the later stages.

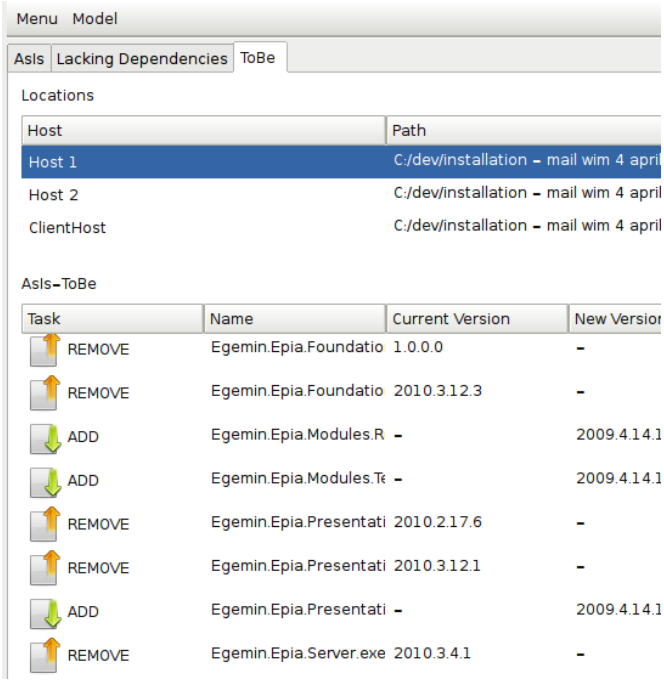

Fig. 6. Evolution viewpoint - installation inconsistency view

The architecture of the SDG solution is under development. For now the part of variability was identified and the domain model is being codified.

VII. Planned project evaluation

Evaluation is crucial to acquire scientific evidence of the validity and usability of the proposed research results. This research will include evaluation of intermediate results as well as validation of the final results.

A. *Sub-goals evaluation.*

After accomplishment of each sub-goal, an internal evaluation (research setting) as well as an external evaluation (setting with domain experts) is planned. The evaluation consists of three activities: planning, evaluation, and results interpretation with feedback. Planning activities include: (1) the description of the experimental context; (2) description of the experimental design; (3) definition of the data collection; (4) determining the approach for analyzing the results; (5) determining the

[2]www.eclipse.org/emf

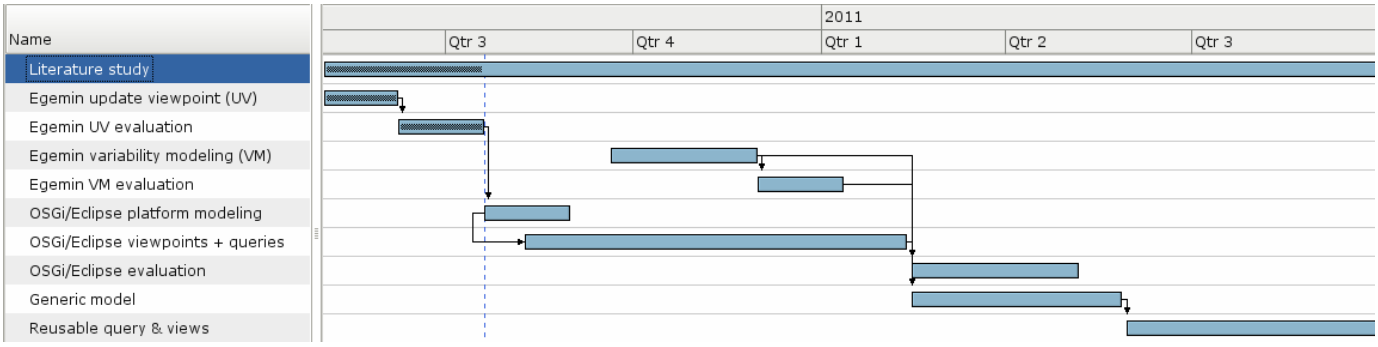

Fig. 5. Planning of the research for years 2010-2011. Tasks are described in context of application domains.

approach to present the results; (6) determining the approach to interpret the results. The actual evaluation will follow the planning. The main goal of the evaluation of intermediate results is to get feedback about intermediate research results and adjust the research direction, which is part of the result interpretation with feedback activity.

B. Validation of the research outcomes.

To validate the AQL Framework controlled experiments will be set up. The primary subjects of the experiments will be architects involved in the industrial cases. The secondary subjects of the experiment will be experienced software engineers, in casu, advanced master students. The definition of appropriate hypothesizes for the experiments will be crucial for the validity of the empirical studies.

VIII. State of the art

Since the early 90's multiple models for architecture descriptions (AD) have been proposed [26]. An AD captures architecturally relevant information of a software system. Well-known viewpoint-centric frameworks, such as [18] and [10], help to express modular, structural, runtime, and deployment views of the systems under development. A viewpoint-based AD serves as a blueprint of the operational system in which the stakeholder requirements are satisfied. Several Architectural Description Languages (ADLs) have been proposed for modeling architectures, such as ACME [14], xADL [11], AADL [13]. However, in practice often UML is used for describing software architectures, despite its limitations [23]. Defining a software architecture can be treated as a problem solving activity [4]. Requirements defines constrains on the problem to solve and the architect is expected to address them with a set of the design decision. Therefore, architecture can be seen as a set of design decisions [17]. Our approach is conform to the meta-model described in the ISO/IEC 42010 standard [16]. The standard proposes an architecture meta-model in which viewpoints and design decisions are expressed. The contribution of this research in the mentioned areas is twofold. Firstly, we aim to define an integrated architecture model that supports multiple views which are defined by different viewpoints representing different stakeholder concerns. Secondly, the architecture repository, which reifies the architectural model, provides an inter-operability layer between stakeholder viewpoints and domain-specific architectural artifacts.

Various approaches for variability modeling have been proposed. Integrated Variability Modelling (IVM) (variability is interwoven with the software artifacts) and Orthogonal Variability Modeling (OVM) (variability is modeled separately) are two different approaches for variability modeling. An example of OVM, based on Variability Point and Variant, is described in [27]. Other authors consider variability modeling from an extensional versus intensional perspective [26]. The extensional approach considers a SPL architecture as a "configurable architecture" (i.e. KOALA [2], COVAMOF [30], and xADL [11]). Intensional approaches capture variability as a change set to given base architecture. Examples of intensional approaches are [15], [21]. In this research, we will realize a concrete OVM approach. This approach is used to separate feature model from a base architecture. Therefore, separation of concerns and complexity management is achieved. However, the integrated variability view on a given part of the system can be achieved by definition of appropriate viewpoint.

Relations between architectural views are important for consistency checking, composition, tracing and model transformation [6]. The AMPLE Traceability Framework [1] is an example of model-driven approach to SPL traceability. In this framework four dimensions of traceability (refinement, similarity, variability, and versioning) are supported. DUALLY [22] supports meta-model level consistency and introduces tools for interoperability between different ADLs, using model transformation mechanisms. [5] proposes relations and compositions as first-class constructs in an ADL. These approaches provide a foundation on which we build our work.

View-based architecture knowledge and view relationships is one of the use case scenarios defined for architectural knowledge management (AKM) [19]. During the last years a number of AKM tools have been developed to support this scenario. Approaches based on the use of keyword-base query engines are SEI-ADWiki [3], ADkwik [29], EAGLE [12], and SEURAT [8]. Approaches based on query languages are ADDSS [9] (with a predefined set of queries), RDF query

language in Knowledge Architect [20], and ontology database Sesame with the SeRQL [7]. Support for stakeholder defined view on the architecture with existing tools is very limited. This research contributes to this with the introduction and realization of the Dynamic Viewpoints concept supported by an AQL.

IX. Conclusion

The motivation for the research proposed in this paper is the need for flexible generation of views of SPL architectures, tailored to the particular needs of the stakeholders. To tackle this problem, an Architecture Query Language Framework is proposed. This framework comprises (1) an architecture model with a supporting repository allowing modeling and reification of domain-specific SPL architectures, (2) Dynamic Viewpoints supporting on-demand view derivation from the model repository according to the stakeholder needs, and (3) an Architecture Query Language and an accompanying query engine which allows flexible and fine-grained querying of the model repository to derive on demand views according to a give viewpoint definition. The research will be evaluated in an empirical study in the context of master course on advanced software architecture, and validated on two cases studies with industrial partners.

Acknowledgment

This research is partially funded by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, the Research Fund K.U.Leuven, and by the Research Foundation Flanders (FWO). Thanks to Nelis Boucké and Alexander Helleboogh for the fruitful discussions and feedback on earlier versions of this paper.

References

[1] N. Anquetil, U. Kulesza, R. Mitschke, A. Moreira, J.C. Royer, A. Rummler, and A. Sousa. A model-driven traceability framework for software product lines. *Software and Systems Modeling*, pages 1–25, 2009.

[2] T. Asikainen, T. Soininen, and T. Männistö. A koala-based approach for modelling and deploying configurable software product families. In *Software Product-Family Engineering*, volume 3014 of *Lecture Notes in Computer Science*, pages 225–249. Springer Berlin / Heidelberg, 2004.

[3] F. Bachmann and P. Merson. Experience using the web-based tool wiki for architecture documentation. Technical report, NASA Center for AeroSpace Information, 7121 Standard Dr, Hanover, Maryland, 21076-1320, USA, 2005.

[4] L. Bass, P. Clements, and R. Kazman. *Software architecture in practice*. Addison-Wesley Professional, 2003.

[5] N. Boucké and T. Holvoet. View composition in multi-agent architectures. Special issue on Multiagent systems and software architecture. *International Journal of Agent-Oriented Software Engineering (IJAOSE)*, 2(2):3–33, 2008.

[6] N. Boucké, D. Weyns, R. Hilliard, T. Holvoet, and A. Helleboogh. Characterizing relations between architectural views. In *Software Architecture*, Lecture Notes in Computer Science, chapter 7, pages 66–81. 2008.

[7] J. Broekstra, A. Kampman, and F. Van Harmelen. Sesame: A generic architecture for storing and querying rdf and rdf schema. *The Semantic Web—ISWC 2002*, pages 54–68, 2002.

[8] J. Burge and DC Brown. An integrated approach for software design checking using rationale. *Design Computing and Cognition*, 4:557–576, 2004.

[9] R. Capilla, F. Nava, S. Pérez, and J.C. Dueñas. A web-based tool for managing architectural design decisions. *ACM SIGSOFT software engineering notes*, 31(5):4, 2006.

[10] D.P. Clements. *Documenting Software Architectures: views and beyond*. Addison-Wesley Professional, 2002.

[11] E. Dashofy, A. van der Hoek, and R. Taylor. A comprehensive approach for the development of modular software architecture description languages. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 14(2):199–245, 2005.

[12] R. Farenhorst and H. van Vliet. Experiences with a wiki to support architectural knowledge sharing. In *3rd Workshop on Wikis for Software Engineering (Wikis4SE)*, 2008.

[13] P.H. Feiler, B. Lewis, and S. Vestal. The SAE Architecture Analysis and Design Language (AADL) Standard: A basis for model-based architecture-driven embedded systems engineering. In *proceeding of the RTAS 2003 Workshop on Model-Driven Embedded Systems, Washington, DC, États-Unis*, 2003.

[14] D. Garlan, R.T. Monroe, and D. Wile. ACME: Architectural description of component-based systems. In *Foundations of Component-Based Systems*. Cambridge University Press, 2000.

[15] Scott A. Hendrickson and Andre van der Hoek. Modeling product line architectures through change sets and relationships. In *ICSE '07: Proceedings of the 29th international conference on Software Engineering*, pages 189–198, Washington, DC, USA, 2007. IEEE Computer Society.

[16] ISO/IEC. *ISO/IEC 42010 (IEEE P42010), Systems and Software Engineering – Architecture Description (WD3)*, 2008.

[17] A. Jansen and J. Bosch. Software architecture as a set of architectural design decisions. In *WICSA '05: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, pages 109–120, Washington, DC, USA, 2005. IEEE Computer Society.

[18] P. Kruchten. The 4+ 1 View Model of architecture. *Software, IEEE*, 12(6):42–50, 1995.

[19] P. Liang and P. Avgeriou. Tools and Technologies for Architecture Knowledge Management. *Software Architecture Knowledge Management*, pages 91–111, 2009.

[20] P. Liang, A. Jansen, and P. Avgeriou. Collaborative software architecting through knowledge sharing. *Collaborative Software Engineering. Springer*, 2009.

[21] N López, R. Casallas, and A. van der Hoek. Issues in mapping change-based product line architectures to configuration management systems. In *Proceedings of the 13th international Software Product Line Conference*, pages 21–30, 2009.

[22] I. Malavolta, H. Muccini, P. Pelliccione, and D. Tamburri. Providing Architectural Languages and Tools Interoperability through Model Transformation Technologies. *IEEE Transactions on Software Engineering*, 36(1):119–140, 2010.

[23] N. Medvidovic, D.S. Rosenblum, D.F. Redmiles, and J.E. Robbins. Modeling software architectures in the Unified Modeling Language. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(1):57, 2002.

[24] B. Michalik and J. Nawrocki. Towards decision centric repository of architectural knowledge. In *CEE-SET'09 conference proceedings*, 2009. to appear in LNCS.

[25] P. O'Leary, R. Rabiser, I. Richardson, and S. Thiel. Important issues and key activities in product derivation: experiences from two independent research projects. In *Proceedings of the 13th International Software Product Line Conference*, pages 121–130. Carnegie Mellon University, 2009.

[26] D.E. Perry and A.L. Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4):40–52, 1992.

[27] K. Pohl, G. Böckle, and F. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 2005.

[28] K. Schmid and M. Verlage. The economic impact of product line adoption and evolution. *IEEE software*, pages 50–57, 2002.

[29] N. Schuster, O. Zimmermann, and C. Pautasso. ADkwik: Web 2.0 collaboration system for architectural decision engineering. In *Proceedings of the Nineteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'2007)*, 2007.

[30] M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch. Covamof: A framework for modeling variability in software product families. *Lecture Notes in Computer Science*, 3154:197–213, 2004.

[31] S. Thiel and A. Hein. Systematic integration of variability into product line architecture design. *Software Product Lines*, pages 67–102, 2002.